BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images may include (but are not limited to):

- BLACK BORDERS
- TEXT CUT OFF AT TOP, BOTTOM OR SIDES
- FADED TEXT
- ILLEGIBLE TEXT
- SKEWED/SLANTED IMAGES
- COLORED PHOTOS
- BLACK OR VERY BLACK AND WHITE DARK PHOTOS
- GRAY SCALE DOCUMENTS

IMAGES ARE BEST AVAILABLE COPY.

As rescanning documents *will not* correct images, please do not report the images to the Image Problem Mailbox.

Coordinating with Obligations

Mihai Barbuceanu
Enterprise Integration Laboratory
University of Toronto
4 Taddle Creek Road, Rosebrugh Building,
Toronto, Ontario, Canada, M5S 3G9
mihai@ie.utoronto.ca

Tom Gray and Serge Mankovski
Mitel Corporation
350 Legget Drive, P.O. Box 13089
Kanata, Ontario, Canada K2K 1X3
{tom_gray,serge_mankowski}@mitel.com

September 30, 1997

Abstract

We explore the view that coordinated behavior is driven by the social constraints (e.g. obligations and interdictions) that agents in organizations are subject to. In this framework, agents adopt those goals that are requested by their obligations, knowing that not fulfilling obligations induces a price to pay or a loss of utility. Based on this idea we build an integrated agent coordination system where we represent the organization, the roles played by agents, the obligations imposed among roles, the goals and the plans that agents may adopt. To consistently update an agent's obligations and interdictions we present an incremental propagation method that also helps with detecting and deciding between conflicting obligations. One strength of this method is that it supports a generic coordination method based on exchanging and locally propagating obligations and interdictions. Once a goal adopted, a special brand of plans, conversation plans, are

available to the agents for effectively carrying out coordinated action. We illustrate the framework with examples from service provisioning in telecommunication networks and supply chain integration.

1 Introduction and Motivation

To build autonomous agents that work coordinately in a dynamically changing world, one of the first things we have to understand is how an agent chooses a particular course of action and how its choices change in face of events happening in the world. In this paper we present a framework that answers this question by construing agents as rational decision makers that exist within organizations. Organizations are systems that constrain the actions of member agents by imposing mutual obligations and interdictions. The association of obligations and interdictions is mediated by the *roles* agents play in the organization. For example, when an agent joins a software production organization in the `system administrator` role, he becomes part of a specific constraining web of mutual obligations, interdictions and permissions - *social constraints or laws* - that link him as a `system administrator` to `developers`, `managers` and every other role and member of the organization. Not fulfilling an obligation or interdiction is sanctioned by paying a cost or by a loss of utility, which allows an agent to apply rational decision making when choosing what to do.

Social laws are objective forces that provide the ultimate motivation for coordinated action at the organization level and to a large extent determine the *mental states* at the individual agent level. Agents "desire" and "intend" the things that are requested by their current obligations, knowing that otherwise there will be a cost to pay. However, current models of collective behavior largely ignore this aspect, trying to explain coordination solely from the perspective of socially unconstrained individuals. For this reason they often impose restrictive conditions that limit the scalability of the models. For example, the Cohen-Levesque account of teamwork [7] requires team members to have the same mutual goal. But this is often not true in organizations where agents have many different and sometimes conflicting goals. As a result, this model can not address coordinated behavior in the presence of multiple and possibly conflicting goals.

To initiate a more realistic investigation of such social constraints and of their role in achieving coordinated behavior, we have built an agent coordina-

tion framework whose building blocks include the entities social constraints are made of: agents, organizations, roles, obligations, interdictions, permissions, goals, constraints, plans, etc. In this framework agents determine what obligations and interdictions currently apply and on this basis decide on their goals, even when these obligations are in contradiction. Once an agent has chosen a goal, it selects a plan to carry it out. Plans are described and executed in a previously reported framework [1] that explicitly represents interactions with other agents by means of structured conversations involving communicative actions. Based on this previous framework, in this paper we present a first semantic and architectural account of representing and reasoning with social constraints represented as obligations, permissions and interdictions (OPI-s) for coordinating agents in organizations. All the reported ideas have been implemented in an extended coordination language and are currently applied (amongst others) to supply chain integration and service provisioning in the telecommunications domain.

2 Organizations and Roles

At the highest level, agents coexist within *organizations* where they play various roles. Our coordination language allows organizations to be described as consisting of a set of roles filled by a number of agents. In the example below, `customer`, `developer`, `help-desk` etc. are roles filled respectively by agents `Customer`, `Bob`, etc.

```
(def-organization O1
  :roles ((customer Customer)
          (developer Bob)
          (help-desk-attendant Bob)
          (development-manager Alice)
          (help-desk-manager John)))
```

An agent can be a member of one or more organizations and in each of them it can play one or more roles. An agent is aware of the existence of some of the other agents, but not necessarily of all of them. Each agent has its local store of beliefs.

A *role* describes a major function together with the obligations, interdictions and permissions attached to it. Roles can be organized hierarchically (for example `developer` and `development-manager` would be both

development-member roles) and subsets of them may be declared as disjoint in that the same agent can not perform them (like help-desk-member and customer). For each role there may be a minimum and a maximum number of agents that can perform it (e.g. minimum and maximum 1 president).

3 Obligation, Interdiction, Permission

An agent a1 in role r1 has an obligation towards an agent a2 in role r2 for achieving a goal G according to some constraint C iff the non-performance by a1 of the required actions allows a2 to apply a sanction to a1. Agent a2 (who has authority) is not necessarily the beneficiary of executing G by the obliged agent (you may be obliged to your manager for helping a colleague), and one may be obliged to oneself (e.g. for the education of one's children).

In our language we provide a construct for defining obligations generically. The generic obligation exists between an :obliged and an :authority agent, when both are in specified roles, for achieving a goal that a third :beneficiary agent needs, whenever a given condition applies. The obligation requires the obliged agent to achieve a goal under given constraints. For example:

```
(def-obligation Reply-to-customer-request
 :obliged help-desk-attendant
 :authority help-desk-manager
 :beneficiary customer
 :condition (received-request
             :from (agent-playing customer)
             :by (agent-playing help-desk-attendant))
 :goal gReply-to-request
 :enforced (max-reply-time 2 hr))
```

The above requires the help-desk-attendant agent (the :obliged party) to reply to a request for support from the customer (the :beneficiary party) under the :authority of the help-desk-manager, in at most two units of time (a constraint on the goal gReply-to-request). This generic obligation becomes active when its condition is satisfied, in this case when the help-desk-attendant receives a request from the customer which requires a reply time not shorter than 2 units of time, because that is the best the

obligation requires the agent to do. When this happens (checked by evaluating the `:condition predicate`), an actual obligation is created linking the `help-desk-attendant` to the `customer` and applying to the actual request received (if many requests are received, as many actual obligations are created). Not shown in the example but allowed by the language are auxiliary conditions and actions that can post arbitrary obligations and interdictions defining the context in which the main obligation has to be carried out. The use of this will be illustrated later on.

In exactly the same manner, our language defines generic and actual *interdictions* (the performance of the goal is sanctioned). *Permissions* (neither the performance nor non-performance are sanctioned) are not represented explicitly. Rather we use a default rule that considers everything not proven to be forbidden as permitted. Finally, the obligations, interdictions and permissions (short OPI-s) of a role are inherited by sub-roles.

Semantics. We model OPI-s using the reduction of deontic logic to dynamic logic due to [11] in a multi-agent framework. Briefly, we define obligation, interdiction and permission as follows, where V_α^{ij} denotes a violation by i of a constraint imposed by j wrt action or goal α (associated with a cost to be paid):

- $F^{ij}\ \alpha \equiv [\alpha]^i V_\alpha^{ij}$: i is forbidden by j to execute α.
- $P^{ij}\ \alpha \equiv \neg F^{ij}\ \alpha$: i is permitted by j to execute α.
- $O^{ij}\ \alpha \equiv F^{ij}(\text{-}\alpha)$: i is obliged by j to execute α.

As shown by [11], this reduction leads to a number of theorems which, as will be shown immediately, allow us to apply a constraint propagation method to infer new OPI-s from given ones in a goal network. The main are as follows (indices dropped for clarity), where ; denotes sequential composition, $\cup$ nondeterministic choice and & parallel composition of actions.

$\models \mathrm{F}(\alpha;\beta) \equiv [\alpha]\mathrm{F}\beta$ (1)
$\models \mathrm{F}(\alpha \cup \beta) \equiv \mathrm{F}\alpha \wedge \mathrm{F}\beta$ (2)
$\models (\mathrm{F}\alpha \vee \mathrm{F}\beta) \supset \mathrm{F}(\alpha \& \beta)$ (3)
$\models \mathrm{O}(\alpha;\beta) \equiv (\mathrm{O}\alpha \wedge [\alpha]\mathrm{O}\beta)$ (4)
$\models (\mathrm{O}\alpha \vee \mathrm{O}\beta) \supset \mathrm{O}(\alpha \cup \beta)$ (5)
$\models \mathrm{O}(\alpha \,\&\, \beta) \equiv (\mathrm{O}\alpha \wedge \mathrm{O}\beta)$ (6)
$\models \mathrm{P}(\alpha;\beta) \equiv < \alpha > \mathrm{P}\beta$ (7)
$\models \mathrm{P}(\alpha \cup \beta) \equiv (\mathrm{P}\alpha \vee \mathrm{P}\beta)$ (8)
$\models \mathrm{P}(\alpha \,\&\, \beta) \supset (\mathrm{P}\alpha \wedge \mathrm{P}\beta)$ (9).

Goals. According to the dynamic logic framework, goals are either `atomic` (non-decomposable), or compositions of type `sequence`, `parallel` or `choice`.

When the type indicates a composition, the goal will specify its components as subgoals. The description of goals also includes the constraints that can be applied to a goal and the optimizations that can be requested. For example:

```
(def-goal gReply-to-request
   :type choice
   :subgoals (gReply-to-request-by-email
              gReply-to-request-by-voice)
   :constraints
     (max-reply-time req-max-reply-time)
   :optimizations (time accuracy))
```

This specifies that goal `gReply-to-request` is of choice type and will be satisfied by executing (at least) one of its two subgoals. It can only be constrained by two constraints, `max-reply-time` and `req-max-reply-time`. Whatever plan is used for this goal, its execution may be optimized for both time and accuracy.

4 Conversation Plans

Finally, agents have plans for achieving their goals. At the outermost level plans specify the goal they can be used for, the constraints they guarantee (a plan for `gReply-to-request` may guarantee `(max-reply-time 2 min)` and thus be applicable for the above obligation) and the optimizations that their execution can provide. A plan is usable for a goal if the requested constraints are satisfied by the plan and preferably (but not necessary) if the plan execution can provide the requested optimizations.

We call plans in our language *conversation plans* because they are descriptions of both how an agent *acts* locally and *interacts* with other agents by means of communicative actions. A conversation plan consists of states (with distinguished initial and final states) and rule governed transitions together with a control mechanism and a local data base that maintains the state of the conversation. The execution state of a conversation plan is maintained in actual conversations. For example, the conversation plan in figure 1 shows how the `Customer` interacts with the `help-desk-attendant` when requesting assistance. After making the request for assistance, the `customer-conversation` goes to state `requested` where it waits for the `help-desk-attendant` to either accept or reject. If the `help-desk-attendant`

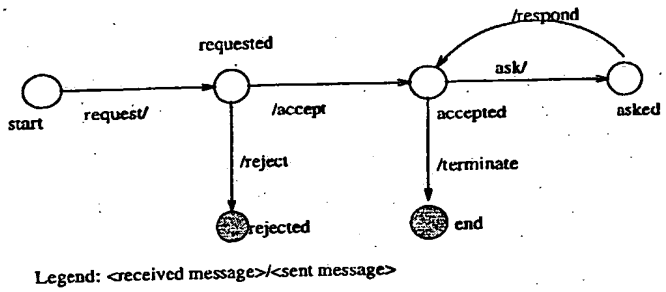

Figure 1: The Customer-conversation

```
(def-conversation-rule 'hda-1
  :current-state 'start
  :received '(request :from (customer ?c)
                      :content ((assistance PBX setup) ??))
  :next-state 'request-received
  :such-that '(help-desk-attendant-available)
  :transmit '(accept :to ?c
                     :conversation ?convn)
  :do '(update-var ?conv '?customer ?c))
```

Figure 2: Conversation rule for help-desk-attendant

accepts to provide assistance, the interaction enters an iterative phase in which the `Customer` asks questions and the `help-desk-attendant` responds. This cycle can end only when the `Customer` decides to terminate it. In each non-final state *conversation rules* specify how the agent interprets incoming messages, how it updates its status and how it responds with outgoing messages. The language in which messages are expressed is a liberal form of KQML [8], but any communicative action language is usable. Figure 2 shows an example of conversation rule that a `help-desk-attendant` would use to respond to a request for assistance.

From the execution viewpoint, each conversation is a *thread*, an agent running all its conversations in parallel, with provisions for suspending a conversation while waiting for other conversations to finish or for events to occur.

Finally, conversation execution can be optimized by dynamically reorder-

ing rules in states to guarantee optimal satisfaction of criteria like *execution time*, *solution accuracy*, *cost*, etc. This is based on a decision theoretic approach that views conversations as Markov decision processes. Full details about this plan execution and optimization framework are available elsewhere [1].

5 Integrating Architecture

The general control architecture integrating the above elements is shown in figure 3. Each agent operates in a loop where:

1. Events (external or internal) are sensed, like the arrival of messages expressing requests from other agents or the occurence of internal conditions that prompt for some action to be taken. In figure 3, we have a message containing a request for assistance from the `Customer` addressed to the `help-desk-attendant`.

2. Sensed events may trigger new obligations and interdictions. These are placed in a newly created workspace called context, each new obligation being also placed in the agent's agenda. Inside each context, the consequences of the triggered obligations and interdictions are inferred by a deontic propagation process described in the next section. During propagation, conflicts between obligations and interdictions are detected and solved as shown in the next subsection.

3. Next, the agent selects an obligation from the agenda. This is either an obligation that has not been tackled before, or one which is under work by the agent. In the former case, the agent looks for a plan that can achieve the goal for which the obligation is formulated and initiates its execution. In the latter case, a plan has already been selected and partially executed, so the agent will just continue execution from where it was left. In the example in figure 3, the `help-desk-attendant` is obliged to respond to `Customer`'s request by activating a plan for the `gReply-to-request` goal that can provide the requested response time.

4. Finally, plan execution takes place in the context created for the triggering event. This context contains specific obligations and interdictions that are enforced. The actions and subgoals involved in the plan are

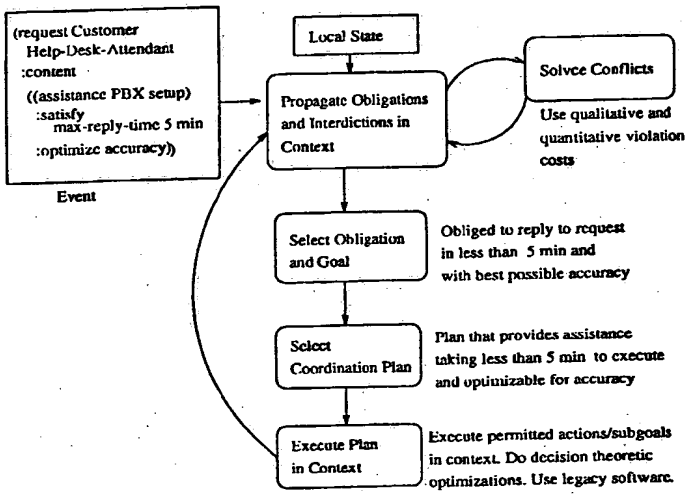

Figure 3: Control Architecture

subject to the constraints this context in that the agent will not execute actions or subgoals that are forbidden in the context, will execute those that are obliged, and will have discretion over those that are permitted but not obliged.

Next, we provide details about these steps.

6 Deontic Constraint Propagation

Each agent's goals form networks in which goals are linked to their subgoals. Figure 4 shows a somewhat arbitrary such network in which g1 is a choice between g2 and g3, g2 is a sequence containing g8, g3 has g4 and g5 executing in parallel, etc. When events and local conditions trigger obligations or interdictions, some of the goals in the network become obliged or forbidden. Because of the existing constraints amongst goals, this may change the deontic status of other goals in the network as well. Deontic constraint propagation is the process by which given some initial deontic labelings of nodes, other implied deontic labelings are inferred. One strength of the dynamic logic reconstruction of deontic logic is that based on theorems like those shown in section 3 we can use a constraint propagation method to infer the

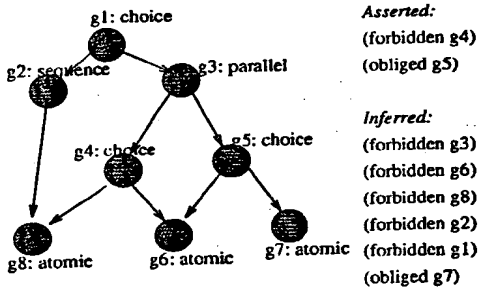

Figure 4: Deontic propagation in a goal network.

deontic labelings implied by an initial set of obligations and interdictions.

Figure 4 illustrates this process using a goal network where we have initially asserted `(forbidden g4)` and `(obliged g5)` in some context `C`. For each of these assertions the propagation process traverses the network along supergoal and subgoal links and applies the deontic theorems listed previously. For example, since g4 is a choice, making it forbidden implies that all its alternatives are also forbidden, according to theorem (2). This makes both g8 and g6 forbidden. Since g8 is forbidden, g2 is also forbidden as a sequence with one action forbidden (1). Propagating along supergoals, g3 becomes forbidden because of g4 (3). Now both g2 and g3 are forbidden, and since they are all the alternatives of g1, g1 becomes forbidden as well (2). When g5 is made obliged, since g6 is forbidden, it follows that g7 must be obliged (derived by working with the dynamic logic definition of obligation and interdiction). All these inferences are added to the `C` context.

Sequential composition of subgoals creates a problem for deontic propagation in that if a (whole) sequence is forbidden (or obliged), the status of subgoals (sequence members) can not be determined in advance. As stated by the previous deontic valitities, only after one subgoal has been executed we can post an interdiction (or obligation) for the remaining part of the sequence. Thus, goal execution has to be monitored and for goals that belong to sequences new propagations need to be performed when these goals are finished.

When goals are asserted as obliged or forbidden, we also allow specifying a cost of violating the obligation or interdiction. This helps us handle conflicting situations. In figure 5 we assert every subgoal of a choice as forbidden with given qualitative costs. Then the choice goal is asserted as forbidden

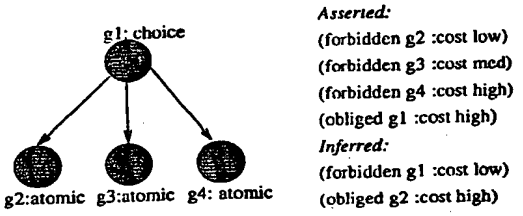

Figure 5: Deontic propagation with costs and conflicts.

with a cost equal to the cost of the smallest cost alternative (this is just one possibility). If later the choice goal is asserted as obliged with a greater cost, then we propagate this upon the smallest cost subgoal. Now we have contrdictory labelings for g1 and g2, but since we also have the violation costs, the agent is justified to do g1 and g2 (accepting their obligatory status) because thus it will incur a smaller penalty. We consider this not as the definitive solution to the problem, but rather as our first shot at it. Note that this scheme works with both *quantitative* and *qualitative* violation costs just as well. The architecture allows each agent to define the nature of violation costs it can handle and its own comparison and decision rules for these.

7 Operation

To see how the architecture operates, assume the `help-desk-attendant` receives a message from the `customer` requesting assistance for some product setup:

```
(request :from (customer ?c)
         :content ((assistance PBX setup)
                   :satisfy (max-reply-time 5 min)
                   :optimize (accuracy)))
```

This matches the `Reply-to-customer-request` obligation of the agent in the `help-desk-attendant` role, placing this obligation on the agenda and asserting its goal, `gReply-to-request`, as obliged in the context created for dealing with this event. Assume further that this goal is a choice between `gReply-to-request-by-voice` and `gReply-to-request-by-email`. The initial obligation, `Reply-to-customer-request`, like any obligation,

can contain predicates checking any relevant conditions and actions asserting other goals as obliged or forbidden in the context of the main obligation. In our case, these conditions also check if the customer is a VIP and look at the constraints and optimizations required. As the customer is a VIP (let's assume) and has stringent response time constraints, the obligation forbids `gReply-to-request-by-email`. Deontic propagation then infers `gReply-to-request-by-voice` as obliged. The system will then select a plan for `gReply-to-request` and attempt to execute it. As this goal is a choice, its execution consists of choosing at least one alternative and recursively executing it. But this choice must not violate the obligations and interdictions that apply in the current context, hence the plan can only select `gReply-to-request-by-voice` and execute it. In general, plans must execute subgoals that are obliged, must not execute forbidden subgoals and have lattitude over permitted subgoals.

Finally, the operational architecture keeps track of how obligations are derived from events like received messages, how goals are derived from obligations and how plans have been selected for goals, using a logical truth maintenance system (LTMS) [10]. This allows agents to retract obligations and interdictions when agents change decisions, like retracting an order or dropping an interdiction because of a more important incompatible obligation. In figure 4 for example, if `(forbidden g4)` is retracted, all inferred assertions will also be retracted as they all rely on this premise. In this organization, contexts are represented as LTMS premise sets. For each obligation in the agenda it tackles, the agent has to switch to different context, which is adequately supported by the LTMS architecture [12].

8 An Application to Feature Interaction

One strength of the approach is the principled way it supports coordination by exchanging and propagating deontic constraints amongst agents. Suppose agent `A` requests something from agent `B`. `A` formulates its request in terms of a set of deontic constraints (obligations and interdictions) that `B` is asked to satisfy. `B` propagates these constraints locally, together with its own relevant constraints, resolving any conflicts according to its own decision procedures and producing in the end a context containing those constraints that it can (or wants to) satisfy. Then `B` satisfies those constraints by executing appropriate plans.

To see a first example of using this in practice, we consider the feature interaction problem in the telecommunications domain [4]. We assume A and B are agents responsible for establishing voice connections amongst their users. The creation and administration of connections can use various levels of functionality, or *features*, that provide different services to subscribers or the telephone administration.

Here are a few examples from the features that are usually available (modern telecommunication services may have many hundreds of features):

- *Incoming Call Screening*: the calee will refuse all calls from callers in an incoming call screening list.
- *Call Forward*: the calee will forward the incoming call to another number.
- *Recall*: if the calee is busy, the caller will be called back later when the calee becomes available.
- *Outgoing Call Screening*: the caller does not allow to be connected to some specified directory numbers.

The feature interaction problem is that often combinations of features interact in undesired ways, affecting the intended functionality of the provided services. For example, several combinations of the above features may interact in an undesired fashion. *Incoming Call Screening* and *Recall* may conflict if *Recall* is done without checking that the number belongs to the incoming call screening list - we shouldn't call back numbers that are not accepted in the first place. Similarly, *Call Forward* and *Outgoing Call Screening* may conflict if a caller is forwarded to a number that it does not wish to be connected to.

The deontic propagation framework can be used to solve such interactions in a principled manner. When agent A wishes to connect to agent B, it will provide to B a set of constraints that correspond to A's relevant features that B must consider. For example, if A has *Outgoing Call Screening*, it will send to B a list of interdictions relative to the numbers that it doesn't allow. If B has *Call Forward*, A's interdictions will be used to forbid forwarding to A's undesired numbers. The propagation process described in the previous section is used as the generic mechanism to solve interactions among features described as deontic constraints.

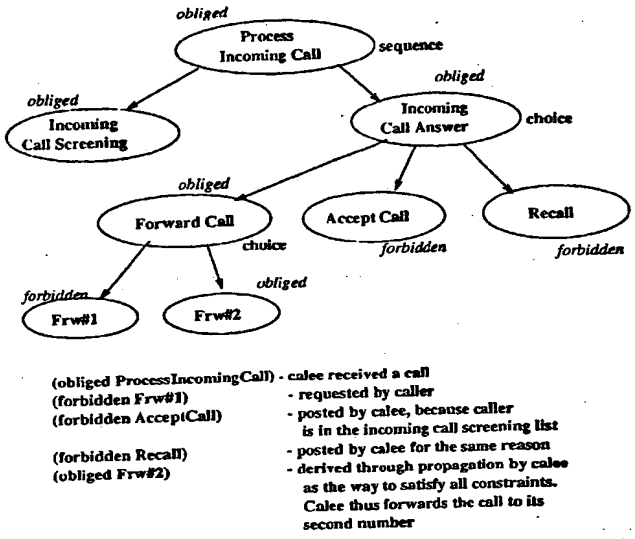

Figure 6: Deontic propagation applied to feature interaction.

For illustration, figure 6 shows the inferences performed by a calee B when receiving a call from A. A has *Outgoing Call Screening* for number #1, and B has *Incoming Call Screening* with A in its incoming call screening list. As the calee, B is obliged to `Process the Incoming Call`. This is a sequence where `Incoming Call Screening` is the first goal. As such it is posted as obliged. When `Incoming Call Screening` is executed, it places an interdiction for `Accept Call` and `Recall` in the current context, because A is on the black list. Then the next subgoal in the sequence, `Incoming Call Answer` is asserted as obliged (see theorem 4). This is a choice with the last two alternatives forbidden. Hence the first alternative, `Forward Call` becomes obliged. But the first number to forward to is forbidden because of caller's request. It follows that forwarding to the second number must be obliged, and this is what the calee, B will do. In conclusion, B does not accept the call and does not set a callback to A if busy. B forwards the call to its second number, because the first is not acceptable to A.

9 Coordinating Teamwork in the Supply Chain

Our second example deals with work coordination in the supply chain of global enterprises. The supply chain of a modern enterprise is a globally extended network of suppliers, factories, warehouses, distribution centers and retailers through which raw materials are acquired, transformed into products, delivered to customers, serviced and enhanced. The key to the efficient operation of such a system is the tight coordination among components. But the dynamics of the enterprise and of the world market make this difficult: customers change or cancel orders, materials do not arrive on time, production facilities fail, workers are ill, etc. causing deviations from plan. Our goal is to achieve coordinated behavior in dynamic systems of this kind by applying our agent coordination technology.

We have built several demonstration systems addressing supply chain issues. In this section we briefly review one of them, focused on modeling team formation, team monitoring and team modification. The supply chain in this case consists of a `Customer` agent, a `Logistics` agent coordinating the joint effort and several plants and transportation agents that participate in the joint work. One typical round of interactions in this setup starts with the `Customer` sending an RFQ (request for quotation, in which an inquiry is made about the cost of an order) to `Logistics`. To answer it, `Logistics` sets up an appropriate run of its (local) scheduling software that decomposes the order into parts doable by the production units in the network and also provides an estimation of whether the order can be executed given the current workload. If the result is positive, `Logistics` tries to obtain tentative agreements from the other production units for executing their part. In this interaction, units are obliged to respond. If the tentative team can be formed, the `Customer` is informed that it can place an order. If this happens, `Logistics` starts another round of interactions in which it asks units to commit to their part of the order. When a unit agrees, it acquires an obligation to execute its part. If everybody agrees, `Logistics` becomes obliged to the `Customer` for execution and the `Customer` to `Logistics` for paying. Then, `Logistics` starts coordinating the actual work by kicking off execution and monitoring its state. Units become obliged to `Logistics` for informing about breakdowns or other events so that `Logistics` can try to replace a unit that can not finish successfully. If breakdowns occur and replacements can not satisfy the initial conditions of the order, `Logistics` tries to negotiate an alternative contract with the `Customer`, e.g. by relaxing some conditions.

We usually run the system with 5-8 agents and about 40-60 actual obligations and conversations each. The specification has about 10-20 generic obligations and conversation plans each with about 200 rules and utility functions. The scheduli ng software is an external process used by agents through an API. All this takes less than 3500 lines of code to describe in our language. We remark the concise ness of the representation given the complexity of the interactions and the fact that the size of this code does not depend on the number of agents and of actua l obligations and conversations, showing the flexibility and adaptability of the representation.

10 Conclusions and Future Work

We believe the major contribution of this work is a unitary coordination framework and language that goes from the fundamental social constraints like obligations and interdictions to the actual structured conversations by which agents directly interact. At the organization level, social constraints are objective forces determining behavior for both human and artificial agents. As such, social constraints are *necessary* components of any account of organizational behavior. At the individual agent level, obligations and interdictions can be viewed as mental states much like beliefs, desires and intentions. Our framework addresses both levels. By describing obligations and interdictions as relations among organization roles, we use them to shape social behavior. By endowing each agent with an internal representation and an engine for reasoning about their obligations and interdictions we effectively integrate obligations and interdictions with the agent's beliefs, goals and plans, extending the BDI model in a new direction. Finally, by providing the technology of deontic constraint propagation we provide a principled approach to "solving" coordination problems, in which agents exchange sets of deontic constraints which they then solve locally according to their own priorities and preferences.

Social constraints have been addressed to some extent previously. [15] describes a theory of coordination within social structures built from roles among which permissions and responsibilities are defined. [14] study the general utility of social laws. [5] stresses the importance of obligations in organizations but does not advance operational architectures. AOP [13] defines obligations locally, but does not really exploit them socially. [9] argues for the necessity of artificial agents with normative positions in today's Internet

world. Finally, [3] uses norms for enhancing the decision making capability of agents, but not for enhancing coordination.

Up to now, our focus has been on prototyping our ideas into systems that can be quickly evaluated and "falsified" in applications. Evaluations based on several supply chain systems (of which we have reviewed one) as well as on service provisioning systems for telecommunications have shown that the coordination language enabled us to quickly prototype the system and build running versions demonstrating the required behavior. Often, an initial (incomplete) version of the system has been built in a few days, enabling us to immediately demonstrate its functionality. Moreover, we have found the approach explainable to and usable by industrial and other engineers interested in their own domain. For example, our latest supply chain system consisting of about 40 agents modeling a realistic enterprise that has several plants, distribution centers and transportation facilities has been developed by an industrial engineer without prior programming experience. In spite of that, a prototype able to simulate the supply chain on a 100-150 weeks horizon during which thousands of plan executions take place has been built in about 3 months.

As current and future work, we are involved in understanding the extent to which our approach to coordination by exchanging and propagating deontic constraints can be used to solve the variety of feature interaction problems classified in the literature [4]. Feature interaction is not only a telecommunications problem, but a problem for service and system creation in general. As such, it is important to investigate general, principled solutions such as our distributed deontic constraint propagation approach.

Another direction is using the representation of obligations and interdictions to build *trusted* and *accountable* agents that provide services in different organizations. Besides actively reasoning about their obligations and interdictions when requested to provide services to internal or external customers, agents also use their representations of obligations to interact with human users in a manner that would prove them trustworthy. We regard this as an important avenue for future research on agents.

11 Acknowledgments

This research is supported, in part, by the Manufacturing Research Corporation of Ontario, Natural Science and Engineering Research Council, Digital

Equipment Corp., Mitel Corp., Micro Electronics and Computer Research Corp., Spar Aerospace, Carnegie Group and Quintus Corp.

References

[1] Barbuceanu, M. and Fox, M. S. 1997. Integrating Communicative Action, Conversations and Decision Theory to Coordinate Agents. Proceedings of Automomous Agents'97, 47-58, Marina Del Rey, February 1997.

[2] Barbuceanu, M. and Fox, M. S. 1996. Capturing and Modeling Coordination Knowledge for Multiagent Systems. *International Journal of Cooperative Information Systems*, Vol.5, Nos. 2 & 3 275-314.

[3] Boman, M. 1997. Norms and Autonomous Artificial Agent Action, submitted (personal communication).

[4] Cameron, E.J., N.D. Griffeth, Y.J. Lin, M.E. Nilson, W.K. Schnure, and H. Velthuijsen. A Feature Interaction Benchmark for for IN and Beyond. In L.G. Bouma and H. Velthuijsen, editors, Feature Interactions in Telecommunication Systems, 1-23, Amsterdam, May 1996. IOS Press.

[5] Castelfranchi, C. 1995. Commitments: From Individual Intentions to Groups and Organizations. In Proceedings of ICMAS-95, AAAI Press, 41-48.

[6] Cohen, P. R. and Levesque, H. 1990. Intention is Choice with Commitment. *Artificial Intelligence* 42, 213-261.

[7] Cohen, P. R. and Levesque, H. 1991. Teamwork. *Nous* 15, 487-512.

[8] Finin, T. et al. 1992. Specification of the KQML Agent Communication Language. The DARPA Knowledge Sharing Initiative, External Interfaces Working Group.

[9] Krogh, K. 1996. The Rights of Agents. In M. Wooldridge,J.P. Muller and M. Tambe (eds) Intelligent Agents II, Agent Theories, Architectures and Languages. *Lecture Notes in AI 1037*, 1-16, Springer Verlag.

[10] McAllester, D. 1980. An Outlook on Truth Maintenance. Memo 551, MIT AI Laboratory.

[11] Meyer, J. J. Ch. 1988. A Different Approach to Deontic Logic: Deontic Logic Viewed as a Variant of Dynamic Logic. *Notre Dame J. of Formal Logic* 29(1) 109-136.

[12] Nayak, P.P. and Williams, B. C. 1997. Fast Context Switching in Real Time Propositional Reasoning. Proceedings of AAAI-97, 50-56, Providence RI.

[13] Shoham, Y. 1993. Agent-Oriented Programming. *Artificial Intelligence* 60, 51-92.

[14] Shoham, Y. and Tennenholtz, M. 1995. On Social Laws for Artificial Agent Societies: Off-line Design. *Artificial Intelligence* 73 231-252.

[15] Werner, E. 1989. Cooperating Agents: A Unified Theory of Communication and Social Structure. In L. Gasser and M.N. Huhns (eds), *Distributed Artificial Intelligence Vol II* 3-36, Pitman.